SAKI SS19 Homework 4

Author: Daniel Seitz

Program code: https://github.com/Anto4ka/Deep-learn-trading

Summary

For a simplified stock market scenario, the challenge was to develop an algorithmic agent that trades for profit. The framework consists of one stock exchange, two types of stock (Company A & Company B) and two expert opinions (Vote A & Vote B). One trade decision a day is required from the trader, which works completely automatic and utilizes Deep-Q-Learning.

Deep-Q-Learning is a model-free reinforcement learning algorithm, whose goal it is to develop a policy, which calculates an (ideally) optimal decision for every possible state of the acting trader. As such it relies on a model-free finite Markov decision process (finite MDP), which is to be presented in the following.

A state is represented by a 4-Tuple (a_1,a_2,b_1,b_2), where $a_1,a_2,b_1,b_2 \in \{1,2,3\}$. a_1 and b_1 both represent the latest development of their respective stock (1: stock value decreased; 2: stock value didn't change; 3: stock value increased), while a_2 and b_2 describe the current expert opinion (1: Sell; 2: Hold; 3: Buy).

An action is represented by a 2-Tuple (x,y) where x is the next action regarding Stock A, while y is the one for Stock B. The Set $\{(x,y) \mid x,y \in \{\text{Buy, Sell}\}\} \cup (\text{Hold, Hold})$ describes the action space.

Using this MDP, a working trader was implemented. Its foundation is based on Q-Learning, which was developed in the late 90s, and deemed useless, as it required enormous amounts of memory and computational power. In its core, Q-Learning tries to calculate the absolute reward of every action (taking as many future steps into account as possible), to find an optimal policy (called the "Q-function"). In this rather modern approach, a deep neural net is used to represent an approximation of the Q-function. It also tries to optimize the outcome of every single day, and does not try to take future steps into account.

In its most simple form the algorithm iteratively executes the optimal action for the current state (predicted by the neural net), to train the model with the input and reward at every single step. In this implementation, multiple additions were included to increase learning speed.

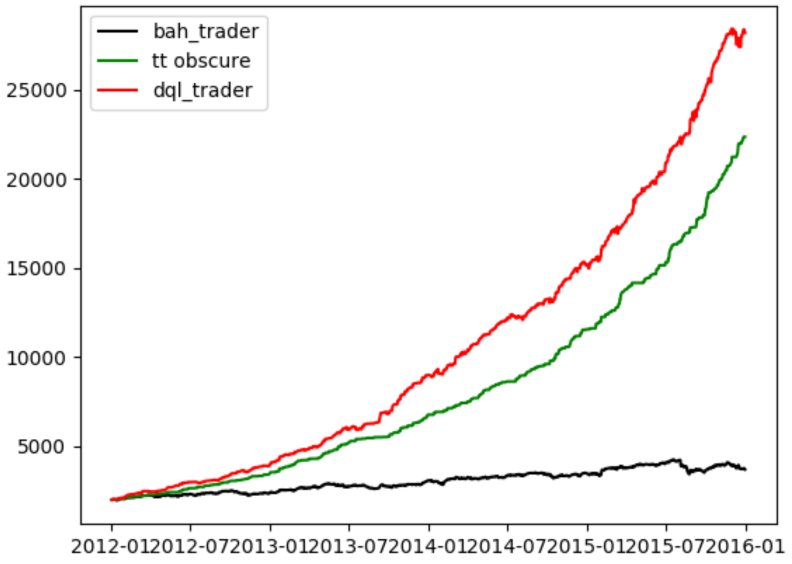
1. Instead of calculating the result of every action a state can proceed with, the Q-function returns all possible results for a state at once. This reduces the processing time excessively.
2. Since a newly initialized neural net only contains meaningless and random values, a variable ε is introduced, to describe the probability of the model just choosing an action by complete chance. While ε starts out as 1.0, it is reduced at every step to mimic the learning process of living organisms. Also, as neural networks are prone to get stuck in local minima, the ε never falls below 0.01.
3. As it is rather inefficient to iteratively fit the model to data of a single action, experience replay is implemented. At every step the state and its result of the chosen action are saved to a list. After a fixed learning phase (e.g 1000 days) the model starts to train, using a batch from the saved memories at random. This also keeps the model from overvaluing older results, which are still affected by a lack of learning.

The neural net itself is a sequential model, containing 2 hidden layers. It expects a state as input, where every element of the 4-tuple is one node. Both hidden layers utilize a Rectified linear unit activation function, with the first layer having 100 Nodes, and the second layer having 50. The final layer represents every actions viability, and as such consists of 4 nodes with a linear activation function.

To calculate the reward of a step, a function was developed which weights the ratio of the old portfolio value and the new value with its general outcome. It also maps some results to fixed values, as to discourage the model from exercising certain behavior (e.g only holding).

Evaluation

With 10 episodes of learning, the model is able to produce a consistently good trader, which beats both the strategy of buy-and-hold and the strategy of always trusting the experts.



Final Values:

- dql_trader: ~28200
- tt_obscure: ~22300
- bah_trader: ~3700

It can also be observed, that the trader completely disregards the (Buy, Buy) and (Hold, Hold) actions during testing.

The absence of (Hold, Hold) could be explained with the fact, that any action containing a Buy could substitute complete Holding, as long as the trader does not have any money at that moment. Preferring a buying action to the holding action could stem from the much higher potential a buying action ensues, hence the better Q-value.

A reason for the model ignoring (Buy, Buy) could be, that the model finds a constant stash of money to be valuable. But since the Q-function is replaced by a neural network, it naturally is very difficult to accurately analyze the reasoning behind a certain behavior.

Possible Improvements

While the model was able to deliver good results, relative to the other traders, its structure is still very unsophisticated.

For example, the state space could be extended to represent the market more accurately. Instead of simply mapping an improvement of stock value to a single integer, one could differentiate between levels of change.

The action space could also be improved in the same way. Instead of using all the money to buy during a single day, one could add actions which only invest a part of the available currency, to improve the handling of situations with different risk levels. For example, a very confident prediction of a stock value rising could be met with full commitment, while shaky predictions result in a very cautious investment.

An other point for possible improvement could be the reward function. A better representation of the relative and absolute portfolio changes could help the model to learn fruitful behavior more quickly.